Contenido

SQL. Consultas básicas

Introducción

En esta unidad, "Lenguaje SQL. Consultas básicas ", daremos los primeros pasos en el conocimiento del lenguaje SQL de consultas.

El DQL (Data Query Language) lo forman las instrucciones capaces de consultar los datos de las tablas.

La introduciremos en los tipos de datos que puede gestionar y en el diseño de consultas sencillas en la base de datos. Después pasaremos a ampliar nuestro conocimiento sobre el lenguaje SQL para aprovechar la potencia que da en el ámbito de la consulta de información. Así, por ejemplo, aprenderemos a ordenar la información, a agruparla efectuando filtrados para reducir el número de resultados, combinar resultados de diferentes consultas Es decir, que este lenguaje es una maravilla que posibilita efectuar cualquier tipo de consulta sobre una base de datos para obtener la información deseada.

Para adquirir un buen conocimiento del lenguaje SQL, es necesario que vaya reproduciendo en el su ordenador todos los ejemplos incorporados en el texto, así como la realización de las actividades y los ejercicios.

La mejor forma de iniciar el estudio del lenguaje SQL es ejecutar consultas sencillas en la base de datos. Antes, sin embargo, nos conviene conocer los orígenes y evolución que ha tenido éste lenguaje, los diferentes tipos de sentencias SQL existentes (subdivisión del lenguaje SQL), así como los diferentes tipos de datos (Números, fechas, cadenas...) que nos podemos encontrar almacenadas en las bases de datos, aunque ya lo vimos en el tema anterior. Y, entonces, ya podremos iniciarnos en la ejecución de consultas simples.

Orígenes y evolución del lenguaje SQL

El modelo relacional en el que se basan los SGBD actuales fue presentado en 1970 por el matemático Edgar Frank Codd, que trabajaba en los laboratorios de investigación de la empresa de informática IBM. Uno de los primeros SGBD relacionales en aparecer fue el System R de IBM, que se va desarrollar como prototipo para probar la funcionalidad del modelo relacional y que iba acompañado del lenguaje SEQUEL (acrónimo de Structured English Query Language) para manipular y acceder a los datos almacenados en el System R. Posteriormente, la palabra SEQUEL se condensó en SQL (acrónimo de SQL).

Una vez comprobada la eficiencia del modelo relacional y del lenguaje SQL, se inició una dura carrera entre distintas marcas comerciales. Así, tenemos:

• IBM comercializa varios productos relacionales con el lenguaje SQL: System / 38 en 1979, SQL/DS en 1981, en DB2 en 1983.

• Relational Software, Inc. (Actualmente, Oracle Corporation) crea su propia versión de SGBD relacional para la Marina de EE.UU., la CIA y otros. El verano de 1979 lanza Oracle V2 (versión 2) para las computadoras VAX (Las grandes competidoras de la época con las ordenadores IBM).

El lenguaje SQL evolucionó (cada marca comercial seguía su propio criterio) hasta que los principales organismos de estandarización intervinieron para obligar a los distintos SGBD relacionales implementar una versión común del lenguaje y, así, en 1986 el ANSI (American National Standards Institute) publica el estándar SQL-86, que en 1987 es ratificado por el ISO (Organización Internacional para la Normalización, o International Organization for Estandarización en inglés).

Diferentes revisiones del estándar SQL que han aparecido desde 1986:

1986 revisión SQL-86 (SQL-87 / SQL1) Publicado por el ANSI en 1986, y ratificado por ISO en 1987.

1989 SQL-89 revisión.

1992 SQL-92 (SQL2) Gran revisión.

1999 SQL:1999 (SQL3) Introducción de consultas recursivas, disparadoras…

2003 SQL:2003 Introducción de XML, funciones Windows...

2006 SQL:2006

Tipo de sentencias SQL

Los SGBD relacionales incorporan el lenguaje SQL para ejecutar distintos tipos de tareas en las bases de datos: definición de datos, consulta de datos, actualización de datos, definición de usuarios, concesión de privilegios... Por este motivo, las sentencias que aporta el lenguaje SQL suelen agruparse en las siguientes:

1. Sentencias destinadas a la definición de los datos (LDD), que permiten definir los objetos (tablas, campos, valores posibles, reglas de integridad referencial, restricciones...).

2. Sentencias destinadas al control sobre los datos (LCD), que permiten conceder y retirar permisos sobre los distintos objetos de la base de datos.

3. Sentencias destinadas a la consulta de los datos (DQL), que permiten acceder a los datos en manera consulta.

4. Sentencias destinadas a la manipulación de los datos (LMD), que permiten actualizar la base de datos (altas, bajas y modificaciones).

En algunos SGBD no existe distinción entre LC y LMD, y únicamente se habla de LMD para las consultas y actualizaciones. Del mismo modo, a veces se incluyen las sentencias de control (LCD) junto con las de definición de datos (LDD).

No tiene ninguna importancia que se incluyan en un grupo o que sean un grupo propio: es una simple clasificación.

SQL asignado

Las sentencias SQL pueden presentar, sin embargo, una segunda sintaxis, sintaxis alojada, consistente en un conjunto de sentencias que son admitidas dentro de un lenguaje de programación llamado lenguaje anfitrión.

Así, podemos encontrar LC y LMD que pueden alojarse en lenguajes de tercera generación como C, Cobol, Fortran ..., y en lenguajes de cuarta generación. Los SGBD suelen incluir un lenguaje de tercera generación que permite alojar sentencias SQL en pequeñas unidades de programación (funciones o procedimientos). Así, el SGBD Oracle incorpora el lenguaje PL/SQL, el SGBD SQLServer incorpora el lenguaje Transact-SQL, SGBD MySQL 5.x sigue la sintaxis SQL 2003 para la definición de rutinas al igual que el SGBD DB2 de IBM.

Tipo de datos

La evolución anárquica que ha seguido el lenguaje SQL ha hecho que cada SGBD haya tomado sus decisiones en cuanto a los tipos de datos permitidos. Ciertamente, los diferentes estándares SQL que han ido apareciendo han marcado una cierta línea y los SGBD se acercan, pero tampoco pueden dejar de apoyar los tipos de datos que han proporcionado a lo largo de su existencia, ya que existen muchas bases de datos repartidas por el mundo que las utilizan.

De todo esto debemos deducir que, a fin de trabajar con un SGBD, debemos conocer los principales tipos de datos que facilita (numéricas, alfanuméricas, momentos temporales...) y debemos hacerlo lo centrándonos en un SGBD concreto (nuestra elección ha sido MySQL) teniendo en cuenta que la resto de SGBD también incorpora tipos de datos similares y, en caso de tener que trabajar, siempre deberemos echar un vistazo a la documentación que cada SGBD facilita.

Cada valor manejado por un DBMS determinado corresponde a un tipo de datos que asocia un conjunto de propiedades en el valor. Las propiedades asociadas a cada tipo de dato hacen que un SGBD concreto trate de forma diferente los valores de distintos tipos de datos.

En el momento de creación de una tabla, es necesario especificar un tipo de dato para cada una de las columnas. En la creación de una acción o función almacenada en la base de datos, es necesario especificar un tipo de dato para cada argumento. La asignación correcta del tipo de dato es fundamental porque los tipos de datos definen el dominio de valores que cada columna o argumento puede contener. Así, por ejemplo, las columnas de tipo DONA'T no podrán aceptar el valor '30 de febrer 'ni el valor 2 ni la cadena Hola.

Dentro de los tipos de datos básicos, podemos distinguir los siguientes:

• Tipo de datos para gestionar información alfanumérica.

• Tipo de datos para gestionar información numérica.

• Tipo de datos para gestionar momentos temporales (fechas y tiempo).

• Otros tipos de datos.

MySQL es el SGBD con el que se trabaja en estos materiales y el lenguaje SQL de MySQL el descrito.

Puede encontrar más información al respecto en el manual de referencia Capítulo 11. Tipo de columna.

Tipo de datos string

Los tipos de datos string almacenan datos alfanuméricos en el conjunto de caracteres de la base de datos. Estos tipos son menos restrictivos que otros tipos de datos y, en consecuencia, tienen menos propiedades. Así, por ejemplo, las columnas de tipo carácter pueden almacenar valores alfanuméricos -letras y cifras-, pero las columnas de tipo numérico sólo pueden almacenar valores numéricos.

Generalmente (hay otros), nosotros utilizaremos los Siguientes:

El tipo CHAR [(longitud)]

Este tipo especifica una cadena de longitud fija (indicada por longitud) y, por tanto, MySQL asegura que todos los valores almacenados en la columna tienen su longitud especificada. Si se inserta una cadena de longitud más corta, MySQL la rellena con espacios en blanco hasta la longitud indicada. Si se intenta insertar una cadena de longitud más larga, se trunca.

La longitud mínima y por defecto (no es obligatoria) para una columna de tipo CHAR es de 1 carácter, siendo la longitud máxima permitida de 255 caracteres.

Para indicar la longitud, es necesario especificarla con un número entre paréntesis, que indica el número de caracteres, que tendrá el string . Por ejemplo: CHAR (10).

El tipo VARCHAR (longitud)

Este tipo especifica una cadena de longitud variable que puede ser, como máximo, la indicada por longitud, valor que es obligatorio introducir.

Los valores de tipo VARCHAR almacenan el valor exacto que indica el usuario sin añadir espacios en blanco. Si se intenta insertar una cadena de longitud más larga, VARCHAR devuelve un error.

La longitud máxima de este tipo de datos es de 65.535 caracteres. La longitud puede indicarse con un número, que indica el número de caracteres máximo que contendrá el string . Por ejemplo: VARCHAR (10).

Tipos de datos numéricos MySQL
soporta todos los tipos de datos numéricos de SQL estándar.

Podríamos realizar una clasificación de los tipos numéricos entre ENTEROS y REALES. Hay diferentes tipos en función de los valores admitidos (puedes consultar el manual). Para simplificar, nosotros utilizaremos los siguientes:

Los tipos de datos INTEGER

El tipo INTEGER (comúnmente abreviado como INT) almacena valores enteros.

Existen varios subtipos de enteros en función de los valores admitidos.

Los tipos de datos enteros admiten la especificación del número de dígitos a mostrar de un valor concreto, utilizando la sintaxis: INT (N), siendo N el número de dígitos visibles.

Así pues, si se especifica una columna de tipo INT (4), en el momento de seleccionar un valor en concreto, se mostrarán sólo 4 dígitos. Hay que tener en cuenta que esta especificación no condiciona el valor almacenado, tan sólo fija el valor a mostrar.

Tipos FLOTANTE, REAL y DOBLE

FLOAT, REAL y DOUBLE son los tipos de datos numéricos que almacenan valores numéricos reales (es decir, que admiten decimales).

Los tipos FLOAT y REAL almacenan en 4 bytes y los DOUBLE en 8 bytes.

Los tipos FLOAT, REAL o DOUBLE PRECISION admiten que especifican los dígitos de la parte entera (E) y los dígitos de la parte decimal (D, que pueden ser 30 como máximo, y nunca mayores que E-2). La sintaxis para esta especificación sería:

• FLOTADOR (E, D)

• REAL (D, A)

• DOBLE PRECISIÓN (E, D)

Tipo de datos DECIMAL y NUMERIC

DECIMAL y NUMERIC son los tipos de datos reales de punto fijo que admite MySQL.

Son sinónimos y, por tanto, se pueden utilizar indistintamente.

Los valores en punto fijo no se almacenarán nunca de forma redondeada, es decir, que si es necesario almacenar un valor en un espacio que no es adecuado, emitirá un error.

Este tipo de datos permite asegurar que el valor es exactamente el que se ha introducido. No se ha redondeado. Por tanto, se trata de una especie de datos muy adecuado para representar valores monetarios, por ejemplo.

Todos dos tipos de datos permiten especificar el total de dígitos(T) y la cantidad de dígitos decimales (D), con la siguiente sintaxis:

Decimales (T, D)

NUMÉRICO (T, D)

Tipo de datos para momentos temporales (fechas)

El tipo de datos que MySQL dispone para almacenar datos que indican momentos temporales son varios. Los más utilizados son los Siguientes:

El tipus de dada DATE

DATE permite almacenar fechas. El formato de una fecha en MySQL es 'AAAAMM-DD', en el que AAAA indica el año expresado en cuatro dígitos, MM indica el mes expresado en dos dígitos y DD indica el día expresado en dos dígitos.

La fecha mínima soportada por el sistema es '1000-01-01'. Y la fecha máxima admisible en MySQL es '31-12-9999'.

El tipo de datos DATETIME

DATATIME es un tipo de dato que permite almacenar combinaciones de días y horas.

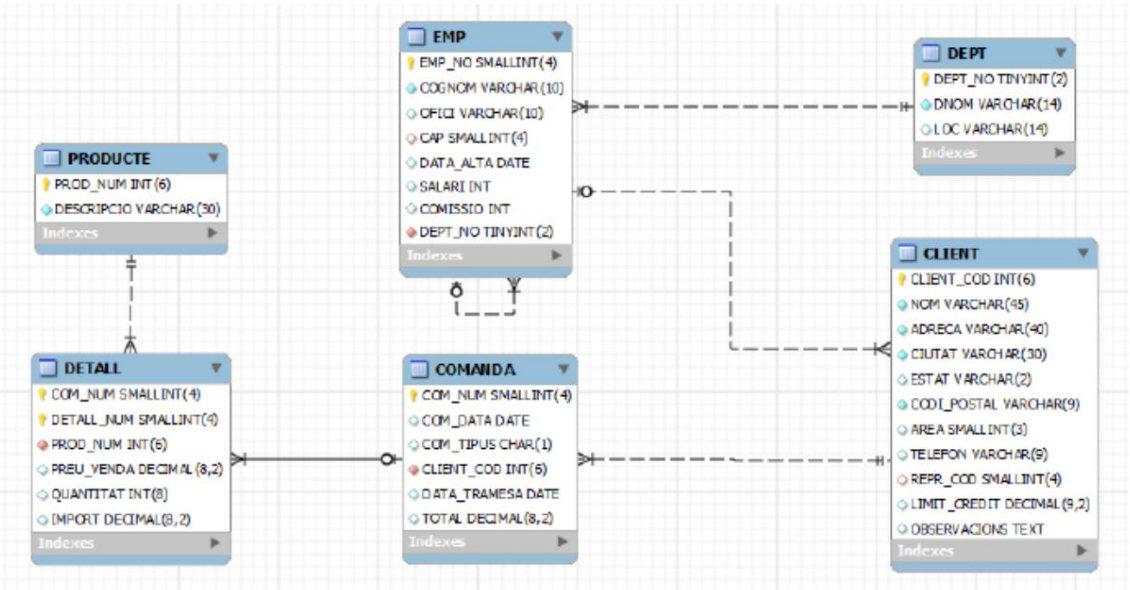
El format de DATETIME en MySQL és 'AAAA-MM-DD HH: MM: SS', en la qual AAAA-MM-DD és el año, mes y día, y HH: MM: SS indican la hora, minuto y segundo, expresados en dos dígitos, separados por ':'.

Consultas simples

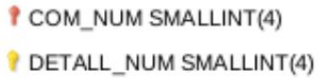
Una vez ya conocemos los diferentes tipos de datos que nos podemos encontrar almacenados en una base de datos (nosotros nos hemos centrado en MySQL, pero en el resto de SGBD es similar), estamos en condiciones de iniciar la explotación de la base de datos, es decir, de empezar la gestión de los datos. Evidentemente, para poder gestionar datos, previamente debe haberse definido las tablas que deben contener los datos, y para poder consultar datos hay que introducirlos antes.
El aprendizaje del lenguaje SQL se efectúa, sin embargo, en sentido inverso; es decir, empezaremos conociendo las posibilidades de consulta de datos sobre tablas ya creadas y con datos ya introducidas. Necesitamos, sin embargo, conocer la estructura de las tablas que deben gestionar y las relaciones existentes entre ellas.

Por ejemplo, la figura 1.1 muestra el diseño del esquema Empresa, implementado con la utilidad modeling del software MySQL Workbench, que utiliza una notación muy intuitiva:

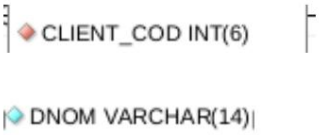
FIGURA 1.1.



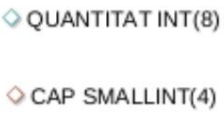
• Las claves primarias indican con el símbolo de una clave.



• Los atributos que no admiten valores nulos van precedidos de un símbolo totalmente coloreado.



• Los atributos que admiten valor nulo van precedidos de un símbolo que está marcado solo su contorno.



• Los atributos que forman parte de una Llave Aliena llevan el símbolo en rojo.

CAP: atributo no obligatorio que forma parte de una clave externa

CLIENT_COD: atributo obligatorio que forma parte de una clave externa

Atributo COD_NUM que forma parte de una Clave Foránea y la Clave Primario

• Las interrelaciones 1: N indican con una línea que termina con tres ramas junto a la entidad interrelacionada en el lado N.

• La opcionalidad se indica con un círculo junto a la entidad opcional y la obligatoriedad con una pequeña línea perpendicular a la interrelación.

• Si coloca el ratón sobre una de las líneas, marcará la Clave Primaria de una de las tablas y su correspondiente Clave Aliena en la otra.

Sentencia SELECT (Consulta)

Tal como lo indica su nombre, esta sentencia permite seleccionar el el usuario pide, no debe indicar dónde debe ir a buscarlo ni cómo debe hacerlo.

La sentencia SELECT consta de distintos apartados que suelen denominarse cláusulas. Dos de estos apartados son siempre obligatorios (SELECT y FROM) y son los primeros que presentaremos. El resto de cláusulas deben utilizarse según los resultados que se quieran obtener.

SQL. Consultas básicas

Símbolos utilizados para especificar la sintaxis en SQL:

[ ] (claudatores). Los corchetes sirven para cerrar texto que no es obligatorio en la instrucción.
Es decir, para indicar parte opcional de la instrucción.

| (barra vertical). Este símbolo (|), la barra vertical, indica opción disyuntiva. Cuando existen diferentes palabras o secciones en la instrucción separadas por la barra, se está indicando que sólo podremos elegir una de las opciones (son opciones, por tanto, excluyentes).

... (puntos suspensivos) Indica que la sección anterior a los puntos suspensivos, se puede repetir una y otra vez.

{} (claves) Las claves sirven para indicaciones obligatorias. Normalmente se utiliza con la barra vertical para indicar que sólo se puede elegir una opción, pero es obligatorio elegir una.

Cláusulas SELECT i FROM

La cláusula SELECT permite elegir columnas y/o valores derivados de éstas (resultados de las expresiones).

La cláusula FROM permite especificar las tablas (también podrían ser vistas: view) en las que es necesario ir a buscar las columnas o sobre las que se calcularán los valores resultantes de las expresiones.

Una sentencia SQL se puede escribir en una única línea, pero para hacer la sentencia más legible suelen utilizar diferentes líneas para las distintas cláusulas.

La sintaxis más simple de la sentencia SELECT utiliza estas dos cláusulas de forma obligatoria:

```
SELECT {* | {[ DISTINCT ] columna | expresión [[AS] alias ], ...}}
FROM nom_taula
[ORDENAR POR {columna [ASC | DESC]} [, columna [ASC | DESC]]]…;
```

En:

*. El asterisco significa que se seleccionan todas las columnas.

DISTINCT hace que no se muestran proyecciones duplicadas afecta a todo lo que hay detrás. Sólo se puede usar una vez inmediatamente después de SELECT

columna. Es el nombre de una columna de la tabla que se desea mostrar.

expresión. Una expresión válida SQL.

alias. Es un nombre que se le da a la cabecera de la columna en el resultado de esa instrucción.
Es pot usar AS o no usar-se

Expresión: una expresión válida en SQL es uno: literal o un valor derivado de aplicar operadores o funciones en las que pueden participar diferentes columnas o valores literales.

Se pueden utilizar operadores matemáticos como: -, +, *, /, DIV, MOD

Y también funciones. Éstas se dividen en bloques según sean sus parámetros o el valor que devuelven. Así tenemos:

- Funciones para cadenas de caracteres (Apartado 12.3. del manual)
- Funciones numéricas (Apartado 12.4. del manual)
- Funciones de fecha y hora (Apartado 12.5. del manual)

Por ejemplo (Ejemplos sobre la BD EMPRESA):

```
SELECCIONAR *
FROM PRODUCTO
```

Proyecta todas las columnas de la tabla PRODUCTO

```
SELECT UPPER(COGNOM), SALARI, SALARI/14 'SALARI MESUAL'
DESDE EMP
```

Proyecta 3 columnas: la primera es el resultado de aplicar una función (UPPER), la segunda es un campo puro y la tercera el resultado de un cálculo. La 3a columna lleva un sobrenombre para la cabecera 'SALARI MESUAL' (si el alias tiene espacios en blanco es necesario ponerlo entre comillas simples).

El lenguaje SQL permite dar un nombre alternativo (alias) a cada columna. Hay que tener en cuenta lo siguiente:

Si el alias está formado por varias cadenas (hay espacios en blanco), es necesario ponerlo entre comillas.

Puede llevar o no la palabra reservada AS.

La cláusula ORDER BY

La cláusula order by permite ordenar el resultado de la consulta.

Siempre va al término de la consulta.

Pueden especificarse varios criterios. El orden de los criterios determinará la ordenación: se aplica el primer criterio y dentro del primero el segundo….

El orden predeterminado es ascendente (ASC)

```
ORDENAR POR {columna [ASC | DESC]} [, columna [ASC | DESC]]…
```

Por ejemplo:

```
SELECT COGNOM, SALARI

DESDE EMP

ordenar por SALARI DESC, COGNOM
```

Los criterios de ordenación no tienen por qué estar definidos sobre las proyecciones del SELECT; pueden estar definidos sobre cualquier columna. Sin embargo, si están definidos sobre columnas proyectadas en el SELECT, podemos hacer referencia a las mismas por el número de orden en el que se proyectos.

La consulta anterior equivaldría a esta:

```
SELECT COGNOM, SALARI

DESDE EMP

ordenar por 2 DESC, 1
```

cláusula donde

La cláusula WHERE se añade detrás de la cláusula FROM con lo que ampliamos la sintaxis de la sentencia SELECT:

```
select expresión|columna [, expresión|columna]...

de taula

[where <condición>];
```

La cláusula WHERE permite establecer los criterios de selección sobre las filas generadas por la cláusula FROM.

Una condición es una expresión que se evalúa a verdadera o falsa para cada una de las filas. Para generar condiciones podemos utilizar, entre otros, los siguientes operadores de comparación:

Operador
>
<
>=
<=
=
<>
!=

Se pueden utilizar tanto para comparar números con per comparando textos y fechas.

En el caso de los textos, las comparaciones se hacen en orden alfabético. Sólo que es un orden alfabético estricto. Es decir, el orden de los caracteres en la tabla de códigos. Así la letra Ñ y las vocales acentuadas nunca quedan bien ordenadas ya que figuran con códigos más altos. Las mayúsculas figuran antes que las minúsculas (la letra “Z” es menor que la "a"). Por lo que para cadenas prácticamente sólo utilizaremos los comparadores de igualdad o diferente.

ENTRE valor1 Y valor2

El operador BETWEEN nos permite obtener datos que se encuentran entre dos determinados valores (incluyendo ambos extremos).

Puede usarse para cualquier tipo de datos aunque para cadenas no tiene mucho sentido. La utilizaremos sólo para expresiones numéricas y fechas.

El orden de los valores es importante: debe ponerse primero el menor AND el mayor.

Ejemplo:

```
SELECT COGNOM, SALARI
DESDE EMP
DONDE SALARI ESTA ENTRE 100000 Y 200000
```

El operador NOT BETWEEN nos permite obtener los valores menores estrictos que el más pequeño y mayores estrictos que el mayor. Ejemplo:

```
SELECT COGNOM, SALARI
DESDE EMP
DONDE EL SALARIO NO ESTÉ ENTRE 100000 Y 200000
```

EN

El operador IN nos permite obtener filas que tengan en la columna alguien de los valores de la lista:

Ejemplo:

```
SELECCIONAR APELLIDO
DESDE EMP
WHERE OFICIO IN ('VENEDOR', 'EMPLEADO');
```

Por supuesto, también podemos utilizar NOT IN .

COMO

El operador LIKE se usa sobre todo con textos y fechas y permite obtener registros cuyo valor en un campo cumpla una condición textual. LIKE utiliza una cadena que puede contener éstos símbolos formando un patrón:

Símbol	
%	Una sèrie de caràcters qualsevol
_	Un caràcter qualsevol

Si no se utilizan comodines, equivale a = (igual)

También podemos utilizar NOT LIKE. Pruébalo.

```
SELECCIONAR APELLIDO
DESDE EMP
DONDE COGNOM COMO 'A%';
```

```
-- Selección de los empleados cuyo APELLIDO contiene la letra 'R' y termina en 'O
SELECCIONAR APELLIDO
DESDE EMP
DONDE COGNOM COMO '%R%O';
```

ES NULO

La cláusula IS NULL devuelve “verdadero” si una expresión contiene el valor nulo, y “Falso” de lo contrario.

La cláusula IS NOT NULL devuelve “verdadero” si una expresión NO contiene un nulo, y “Falso” en caso contrario.

Ejemplos:

```
-- Devuelve las filas de EMP NO valor en el campo CAP
SELECCIONAR *
DESDE EMP
DONDE CAP ES NULO;
```

```
-- Devuelve a los empleados que SÍ QUE tienen ningún
SELECCIONAR *
DESDE EMP
DONDE CAP NO ES NULO;
```

Operadores lógicos

Podemos realizar condiciones complejas a partir de condiciones simples enlazándolas con operadores lógicos.

Operador	
AND	Retorna VERTADER si les dues expressions són vertaderes
OR	Retorna VERTADER si almenys una de les expressions és vertadera
NOT	Inverteix la lògica de l'expressió que està a la seua dreta. Si l'expressió és vertadera, retorna FALS.

Este ejemplo sería equivalente al de BETWEEN

```
SELECT COGNOM, SALARI
DESDE EMP
DONDE SALARIOS >=100000 Y SALARIOS <=200000
```

Este ejemplo sería equivalente al de NOT BETWEEN

```
SELECT COGNOM, SALARI
DESDE EMP
DONDE SALARIO <=100000 O SALARIO >=200000
```

MUY IMPORTANTE:

La mayor precedencia (lo que primero se 'resuelve') la tiene NOT.

AND tiene precedencia sobre ORO. La precedencia puede modificarse con paréntesis. Es álgebra booleana (de Boole). Así, por la propiedad distributiva, las siguientes expresiones son equivalentes:

```
Condición1 AND Condición2 OR Condición1 AND Condición3 =
= Condición1 AND (Condició2 OR Condición3)
```

Nota: Funcionan como la multiplicación (AND) respecto a la suma (OR)

Las tablas de verdad son las siguientes:

Y

C1	C2	C1 y C2
F	F	F
F	V	F
V	F	F
V	V	V

O

C1	C2	C1 o C2
F	F	F
F	V	V
V	F	V
V	V	V

C1	C2	C3	C1 Y C2 O C3	C1 Y (C2 O C3)
F	F	F	F	F
F	F	V	V	F
F	V	F	F	F
F	V	V	V	F
V	F	F	F	F
V	F	V	V	V
V	V	F	V	V
V	V	V	V	V

Funciones de Agregado (valores resumen)

Ya hemos visto cómo hacer cálculos con los datos de una consulta e incluso cómo utilizar funciones en esos cálculos. Pero hasta ahora las funciones utilizadas sólo podían utilizar información procedente de datos de la misma fila. Es decir, no podíamos sumar, por ejemplo, datos procedentes de diferentes filas. Sin embargo, hacer cálculos con datos de diferentes filas es una necesidad muy habitual.

Las funciones de cálculo con grupos son las encargadas de realizar cálculos en vertical (usando datos de diferentes filas) en lugar de en horizontal (usando datos procedentes de la misma fila en la que vemos el resultado).

Cuando hacemos una consulta de valores resumen el resultado es una ÚNICA HILA Y SOLO PODEMOS PROYECTAR FUNCIONES DE AGREGADO.

Son valores resultado de un conjunto de filas (las que pasan el filtro de WHERE). No podemos en esa misma consulta proyectar valores individuales de filas (aunque la ejecución no de error).

Si lo piensa, es lógico. Por ejemplo: en una misma pregunta (consulta) yo no puedo preguntar cuántos sois en clase y cuál es el apellido. Si estoy consultando valores de un conjunto, no tiene sentido consultar valores individuales.

Las funciones de cálculos de totales para datos agrupados son las siguientes:

Función	Significado
COUNTO(expresión)	Cuenta los elementos (files) de un grupo que tienen valor (no tienen valor NULL) en la expresión. Si lo que queremos es contar las filas de un grupo: Se suele indicar un asterisco (COUNT(*))en lugar de una expresión; la cuenta sería la misma que si ponemos una expresión que tenga valor para todas las filas del grupo Hay que tener en cuenta que esta función ignora los valores nulos a la vez de contar, por lo que la expresión COUNT(telefono) devuelve la cantidad de filas que tienen valor en el campo teléfono. No cuenta una fila si tiene NULL en el campo o expresión correspondiente. Expresión puede ser de CUALQUIER TIPO
SUM(expresión)	Suma los valores de la expresión (solo valores numéricos) Expresión sólo puede ser de TIPO NUMÉRICO
AVG(expresión)	Calcula la media aritmética sobre la expresión indicada (solo valores numéricos) Expresión sólo puede ser de TIPO NUMÉRICO
MIN(expresión)	Mínimo valor que toma la expresión indicada Expresión puede ser de CUALQUIER TIPO
MAX(expresión)	Máximo valor que toma la expresión indicada Expresión puede ser de CUALQUIER TIPO
STDDEV(expresión)	Calcula la desviación estándar (solo valores numéricos) Expresión sólo puede ser de TIPO NUMÉRICO
VARIANCE(expresión)	Calcula la varianza (solo valores numéricos) Expresión sólo puede ser de TIPO NUMÉRICO

Nosotros sólo utilizaremos COUNT, SUM, AVG, MIN y MAX.

Todas ellas requieren trabajar con grupos (más adelante se explica cómo agrupar filas).

Si no se indican grupos, las funciones trabajan sobre todas las filas seleccionadas (el origen de las que son las tablas del FROM y que cumplen las condiciones del WHERE).

SQL. Consultas básicas

Una consulta es como una pregunta y debe tener sentido. Por ejemplo: no tendría sentido preguntar cuántos alumnos hay en el instituto y su apellido; o bien preguntamos por valores individuales de los alumnos (de cada uno de los alumnos) o preguntamos por valores resumen de todo el grupo de alumnos.

> Cuando hacemos una consulta con funciones de agregado obtenemos una sola fila por cada grupo y sólo es posible proyectar funciones de agregado o valores comunes a todo el grupo (criterios de agrupación).

Por ejemplo, si tenemos esta tabla:

Cod_trabajador	Nombre	Salario	Teléfono
1	Pedro	1200	630444444
2	Bien	1500	666666666
3	raquel	1650	615156651
4	Sebastián	980	NULO
5	Marcos	1200	NULO
6	Mónica	1650	600000000
7	Raúl	1200	578964123
8	Mireia	980	635789412
9	Gorka	1650	NULO
10	Maia	1650	555555555

Suponiendo que son los datos de la mesa trabajadores, la consulta:

```
SELECT MAX(salario) FROM trabajadores;
```

Devuelve el valor 1650, que es el máximo salario.

```
SELECT COUNT(*), COUNT(salario) FROM trabajadores;
```

Devolvería 10 y 10, ya que hay 10 trabajadores en la mesa y todas las filas tienen valor en el atributo salario.

```
SELECT COUNT(telefón) FROM trabajadores;
```

Devolvería 7 ya que hay sólo 7 filas en la mesa trabajadores que tienen valor en el atributo telefon

Las consultas anteriores sólo tienen una columna pero podemos proyectar las columnas que deseamos. Por ejemplo:

```
SELECT MAX(salarios), MIN(salarios),COUNT(*),COUNT( salario DISTINCT)
FROM trabajadores;
```

MAX(salarios)	MIN(salarios)	COUNT(*)	COUNT(DISTINCT salario)
1650	980	10	4

Manejo de los valores nulos

Las funciones de cálculo agrupado ignoran los valores NULL.

La función COUNT, se suele usar con asterisco (*) como objeto de cálculo, lo que hace que cuenta filas independientemente de su contenido. Si usamos una expresión, cómo COUNT(telefono), no contaría las filas que tengan un teléfono nulo en ellas.

Uso de DISTINCO en las funciones de totales

Las funciones anteriores admiten que antepongamos el término DISTINCT antes de la expresión. De esa forma, sólo se tienen en cuenta los valores diferentes. Por ejemplo:

```
SELECCIONAR RECUENTO ( salario DISTINTO)
```

El resultado es 4 porque sólo hay 4 salarios distintos. Lo mismo ocurre con el resto de funciones, DISTINCT hace que se ignoren los valores repetidos.